Squeryl

A Scala ORM and DSL for talking to Databases with minimum verbosity and maximum type safety

Who is Jay Taylor?

- Grew up in Palo Alto, California
- Attended university in Utah
- Worked with many different kinds of businesses and startups
- I love learning programming languages
- Scala aficionado for several years now

Relevant experience

Worked with variety of different kinds of startups and more established businesses:

Medical devices
Telecommunications
Marketing and Advertising
Games
Social Media

What is an ORM?

- ORM stands for <u>O</u>bject-<u>R</u>elational <u>M</u>apper

1. Given a database and an interface to programmatically interact with it, there is still a gap between relational databases and object-oriented languages
2. Object-relational mappers serve as a bridge between relational databases and object-oriented programs

What is an ORM?

- Why do we need an ORM? What can they offer?

What is an ORM?

Why is an ORM needed? What’s in it for me?

1. Security: Used properly, ORMs can help avoid database injection vulnerabilities

2. High level of abstraction can be powerful

3. Helpful with regard to DRY principle

Scala persistence libraries

Scala persistence libraries

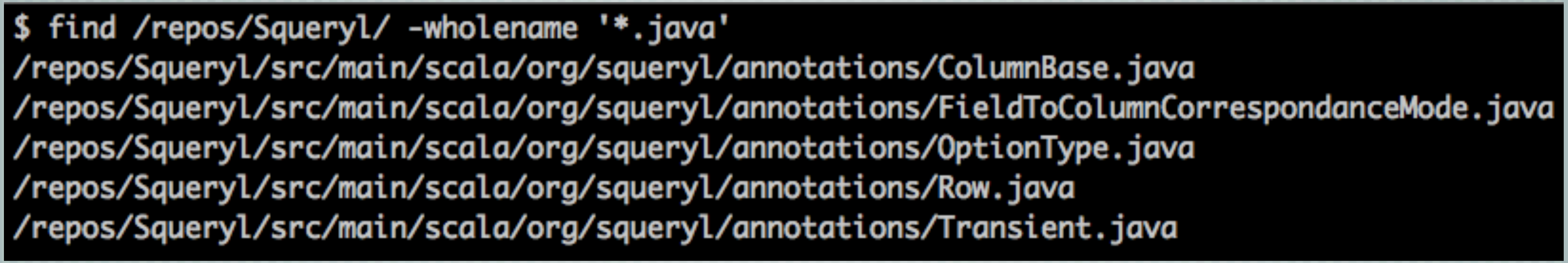
- Hibernate
  - Widely used open-source JPA library

Scala persistence libraries

- The Anti-ORM: Play Framework Anorm
  - ANORM is Not an Object-Relational Mapper
  - No automatic mapping between objects and relational models
  - "SQL is a great DSL for talking to relational databases."

Scala persistence libraries

- Squeryl
  - Developed and maintained by Maxime Lévesque
  - Has been around since early 2010
  - Pure-Scala library
    ..well, almost (annotations still require a bit of Java ;)



```
$ find /repos/Squeryl/ -wholename '*.java'
/repos/Squeryl/src/main/scala/org/squeryl/annotations/ColumnBase.java
/repos/Squeryl/src/main/scala/org/squeryl/annotations/FieldToColumnCorrespondanceMode.java
/repos/Squeryl/src/main/scala/org/squeryl/annotations/OptionType.java
/repos/Squeryl/src/main/scala/org/squeryl/annotations/Row.java
/repos/Squeryl/src/main/scala/org/squeryl/annotations/Transient.java
```

Scala persistence libraries

- Squeryl
  - Mixes the world of ORMs with the world of DSLs

What is “DSL”?

- In this context, not a type of internet service
  - DSL”actually stands for <u>D</u>omain <u>S</u>pecific <u>L</u>anguage

Domain Specific Languages

- DSLs are worth knowing about because they..
  - Take a narrow part of programming and make it more understandable
  - Enable domain experts to understand code

Squeryl

Squeryl

- Attributes unique to Squeryl
  - Type-safe: Query statements are parsed and compiled into bytecode
  - Complete avoidance of fragile string-based query languages

Squeryl

- Community
- Responsive maintainer
- Compatible with most major relational databases
- Explicit control over granularity, eagerness, and laziness
- Composability

WE'LL DO IT
LIVE!

Let's do it live (-ish)

- Let's build a basic bulletin board system with Squeryl

Let’s do it live

- Let’s build a basic bulletin board system with Squeryl

... we’ll call it **Squerbuld**

- *What components will we need?*

Squerbuld

- Will be capable of:
    - Displaying all posts on the main index page
    - Each post links to a view of just that post
    - Create new posts
    - Reply to existing posts

Squerbuld

- Technology Stack:
  - Play Framework 2.0
  - Squeryl 0.9.6
  - Postgresql

Squerbuld

- To use Squeryl, we need to do a few things:
  1. Add Squeryl dependency to SBT project file
  2. Create our schema definition
  3. Create our database model(s)
  4. Provide Squeryl with a database connection
  5. Ensure the database gets created

Squerbuld

1. Add Squeryl dependency to SBT project

Squerbuld

```
import sbt._
import Keys._
import PlayProject._

object ApplicationBuild extends Build {

    val appName         = "SquerylDemo"
    val appVersion      = "1.0-SNAPSHOT"

    val appDependencies = Seq(
      "org.squeryl" %% "squeryl" % "0.9.6.2-SNAPSHOT" withSources(),
      "postgresql" % "postgresql" % "8.4-701.jdbc4"
    )

    val main = PlayProject(appName, appVersion, appDependencies).settings(
      resolvers ++= Seq(
          "Scala.sh Releases" at "http://scala.sh/repositories/releases",
          "Scala.sh Snapshots" at "http://scala.sh/repositories/snapshots"
        )
    )
}
```

Squerbuld

- 2. Create our schema definition

Squerbuld

```
package models

import org.squeryl.Schema
import org.squeryl.PrimitiveTypeMode._


object BulletinBoardSchema extends Schema {

    val posts = table[Post]("Post")

    val postToReplies = oneToManyRelation(posts, posts)
        .via((pA, pB) => pA.id === pB.refPostId)

}
```

Squerbuld

- 3. Create our database model(s)

Squerbuld

```
package models

import org.squeryl._
import org.squeryl.dsl._
import org.squeryl.PrimitiveTypeMode._
import java.sql.Timestamp

case class Post(
    val id: Long = 0L,
    val created: Timestamp,
    val author: String,
    val subject: String,
    val body: String,
    val refPostId: Option[Long] = None
) extends KeyedEntity[Long] {
    lazy val replies: OneToMany[Post] = BulletinBoardSchema.postToReplies.left(this)

    def getReplies: List[Post] =
        inTransaction {
            replies. toList
        }
}
```

Squerbuld

- 4. Provide Squeryl with a database connection

Squerbuld

```
package models

import org.squeryl.{Session, SessionFactory}
import org.squeryl.adapters.PostgreSqlAdapter
import play.Logger
import java.sql.DriverManager

class NewAgePostgreSqlAdapter extends PostgreSqlAdapter {
    override val usePostgresSequenceNamingScheme: Boolean = true
}

object DbPool {
    Class forName "org.postgresql.Driver"

    SessionFactory.concreteFactory = Some(() =>
        Session.create(
            DriverManager.getConnection("jdbc:postgresql://localhost:5432/squeryl"),
            new NewAgePostgreSqlAdapter
        )
    )

    Logger.info("DB Pool initialized")
}
```

Squerbuld

5. Ensure the database gets created

Let's also make sure the database connection is initialized when the application starts

Squerbuld

```
class Global extends GlobalSettings {

    /**
     * Touch lazy objects to wake them up.
     */
    override def onStart(app: Application) {
        import models.DbPool

        // Wake up the lazy object
        DbPool.getClass

        try {

            transaction {
                BulletinBoardSchema.create
            }

        } catch {
            // NB: This will happen everytime after the first time the schema
            // is successfully created
            case e: Exception => Logger.info("Global exception: " + e.getMessage)
        }
    }

}
```

Squerbuld

- Play Framework also requires a few things from us..
  - Controllers
  - Route definitions
  - View templates

Squerbuld

- Controllers
  - Index
  - Post viewer to view individual posts
  - New posting form display
  - New post processor to do the insert

Squerbuld

The index controller is a piece of cake

```
def index = Action {
    Ok(views.html.index(Post.all))
}

// + models.Post companion object with `all` method:

object Post {
    def all: List[Post] = inTransaction {
        from(BulletinBoardSchema.posts)(p => select(p) orderBy(p.created desc)).toList
    }
}
```

Squerbuld

The post viewer controller is also a piece of cake

```
def displayPost(postId: Long) = Action { request =>
    Ok(views.html.displayPost(Post(postId).head))
}

// + models.Post companion object with `apply` method:

object Post {
   def all: List[Post] = inTransaction {
        from(BulletinBoardSchema.posts)(p => select(p) orderBy(p.created desc)).toList
   }

   def apply(id: Long): Option[Post] = inTransaction {
        from(BulletinBoardSchema.posts)(p => where(p.id === id) select(p)).headOption
    }
}
```

Squerbuld

- The new posting form display controller is yet more cake

```
def newPostForm(refPostId: Option[Long] = None) = Action { request =>
    val refPost = refPostId match {
        case Some(id) => Post(id)
        case None => None
    }
    Ok(views.html.newPost(refPost))
}
```

Squerbuld

- The new posting processor controller is the substantial piece

```
def processNewPost = Action { request =>
        val formData = request.body.asFormUrlEncoded.head

        def getParam(name: String): String = {
            val result = formData.get(name).flatMap(seq => seq.headOption)
            result match {
                case Some(value) => value
                case None => throw new Exception("Missing required parameter: " + name)
            }
        }

       val refPostId = formData.get("refPostId").flatMap(seq => seq.headOption) match {
            case Some("") => None
            case None => None
            case Some(value) => Some(value.toLong)
        }

        val message = Post(
            created=new Timestamp(System.currentTimeMillis),
            author=formData.get("author").flatMap(seq =>
                seq.headOption
            ).getOrElse("Anonymous"),
            subject=getParam("subject"),
            body=getParam("body"),
            refPostId=refPostId
        )

        transaction {
            BulletinBoardSchema.posts.insert(message)
        }

        Redirect("/")
    }
```

Squerbuld

The final piece is the routes file

```
# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~~

# Home page
GET     /           controllers.Application.index

GET     /newPost    controllers.Application.newPostForm(refPostId: Option[Long] ?= None)
POST    /newPost    controllers.Application.processNewPost

GET     /view/:postId       controllers.Application.displayPost(postId: Long)
```